An accident or mishap is traditionally defined by engineers as an unplanned event or series of events that leads to an unacceptable loss such as death, injury, illness, damage to or loss of equipment or property, or environmental harm. Accidents usually involve unwanted and unexpected releases of energy or dangerous substances. By this definition, computers or cause physical harm. However, computers can contribute within the context of system safety.

Accidents usually involve unwanted and unexpected releases of energy or dangerous substances. By this definition, computers or cause physical harm. However, computers can contribute within the context of system safety. System-safety engineers define safety in terms of hazards and risk. A hazard is a set of conditions (i.e., a state) that can lead to an accident given certain environmental conditions. Examples of hazards are guard gates not lowering when trains approach traffic crossings, pressure increasing above some threshold in a boiler, or aircraft violating minimum separation standards.

Risk is defined as a function of (1) the likelihood of a hazard occurring, (2) the likelihood that the hazard will lead to an accident, and (3) the worst possible potential loss associated with such an accident. Risk can be reduced by decreasing any or all of these three risk factors. Decisions about the use of safety-critical systems often rest on a final assessment of the risk involved. When a numerical risk assessment is attempted, the numbers are derived from (1) historical information about the reliability of the individual components and the models that define the connection between these components or (2) historical accident data about similar systems.

The first of these procedures works for hardware because historical reliability figures are available for standard parts that have been used for decades. Design errors are usually not considered and the failure probabilities are based solely on random wear-out data. The second assessment approach, i.e., the use of historical accident data, is feasible for physical systems because they tend to change very little in basic design over long periods of time.

Neither of these assumptions is true for software, however. Software is usually specially constructed each time it is used. Even when software is reused, the interfaces between software components are extremely complex in comparison to hardware. This complexity makes it difficult to build a model of component interaction that accurately combines the individual component failure probabilities into an integrated software failure probability. Although such models can be constructed, their predictions are not yet reliable. For newly-constructed software components, measuring reliability during testing and development remains a research topic. Although it may be possible now or soon to derive these numbers for failure probabilities in the mid-ranges, the very low failure probabilities required in safety-critical systems require more experience with the software than could possibly be obtained in a realistic amount of time [7].

Furthermore, failure of the software is not what needs to be considered when evaluating software safety. Software hazards usually involve outputs produced under the wrong circumstances, e.g., directing two planes into the same airspace, providing the wrong information to the nurse in a patient monitoring system, or turning lights green in a railway signaling system when the track is not clear. There is no way to measure the probability that software will not exhibit one particular behavior. A reliability figure for the software is not only not what is needed or appropriate here, it will also be very misleading.

Most software reliability models define failures in terms of deviations from the software requirements specification; most accidents involving software are due to errors in the software requirements specification [1]. Therefore, measuring probability of concurrence of the software with the software reliability specification will not give an adequate estimate of risk. Another problem in evaluating the probability of events caused by design errors is that only one occurrence may be unacceptable when the consequences of that event are very serious. Assuming that something will never happen is not easily demonstrated using probabilistic approaches. There are, however, ways to design software and systems to provide very high confidence that they will not exhibit that behavior or that the system will fail-safe if they do.

As an example, the software errors that killed several people in the Therac-25 incidents involved timing-related race conditions. One error was triggered by input occurring at a certain speed while the other required temporal coincidence between an infrequent software state and a rare external event. There is little likelihood that testing and reliability modeling techniques could have found or predicted these errors. However, the hazard itself, involving exposure to massive amounts of radiation, was easily anticipated. The accidents could have been prevented by either putting a simple check for the hazardous software state (in the form of an IF statement) into the software or by using a hardware interlock or both. System-safety engineers protect against particular design errors by building protective devices to prevent those behaviors and using backup procedures in case those protective devices fail. This is what needs to be emphasized when considering software safety.

A problem with all of the probabilistic models, both for hardware and for software, is that they are usually based on easily-violated assumptions about the operation of the system.
and its components. One of the most comprehensive probabilistic risk analyses that has been performed is a nuclear reactor safety study called Wash-1400 [5,6], which attempted to demonstrate that nuclear power plants have acceptable risk. This study has been criticized [6] for using elementary data that was incomplete or uncertain and for making many unrealistic assumptions. For example, independence of failures was assumed while common mode failures were largely ignored. Additionally, it was assumed that nuclear power plants are built to plan and are properly operated. Recent events suggest that this may not be the case. Critics also maintain that the uncertainties are very large, and therefore the calculated risk numbers are not very accurate. Almost all such probabilistic risk assessments involve equally unrealistic assumptions as do most software assessment techniques.

Non-probabilistic approaches to assurance of software safety, therefore, may be more appropriate given the importance of design errors in software compared to the major emphasis put on random wear-out failures in physical systems. This is true, by the way, for both hardware and software systems. System safety engineers have sometimes concentrated more on getting the proper numbers out of their models than on providing hazard control and management procedures. Furthermore, with the highly complex hardware systems now being built, design errors are assuming more importance and cannot be ignored in risk assessments.

A possible non-probabilistic approach to assurance and evaluation of safety [1,2,3,4] involves augmenting good software engineering practice with: (1) analysis procedures to identify hazards, (2) elimination and control of these hazards through various types of hardware and software interlocks and other protective devices using several layers of protection, (3) application of various types of safety analysis techniques (including formal and informal verification of safety) during the software development to provide confidence in the safety of the software and to aid in the design of hazard protection, and (4) evaluation of the effectiveness of the analysis and design procedures to assess the level of confidence they merit. This latter evaluation might be qualitative or involve prior quantitative evaluation of the procedures through controlled experimentation or extensive industrial experience.

This basic approach can be categorized as providing layers of protection so that errors in the analysis or in the software that could lead to hazards are detected or handled in another way. The safety verification and analysis are backed up by using software safety design techniques that protect against hazardous states that might result from undetected software faults, including those stemming from flaws in the software requirements specification. And both of these need protection that is external to the software.

Will this type of safety analysis suffice to ensure adequate safety in computer-controlled systems? The answer to this question depends on the formality and power of the particular analysis techniques applied, the amount and effectiveness of protection provided to guard against errors in the software and in the analysis, and the level of assurance that is required (which in turn depends upon the level of acceptable risk). There are few systems that are completely free of risk. What is required for a system to be usable is that it have acceptable risk. The level of risk that is acceptable will vary with the type of system and the potential losses that are possible. Acceptable risk for a military fighter aircraft or for an experimental aircraft such as the X-29 where potential loss primarily involves property will be much higher than acceptable risk for a commercial aircraft where public safety is involved. In systems that currently are controlled or protected by non-computerized means, the decision about the introduction of computers may involve a judgement as to whether the resulting risk is increased and how much confidence can be placed on this judgement.

References